Docker : Maîtriser la conteneurisation pour les développeurs et les opérations

Table of Contents

Étude de cas : Rationalisation du flux de travail de développement Web

Imaginez que vous êtes un développeur Web rejoignant fraîchement une équipe d'autres développeurs sur un projet d'application de messagerie fonctionnant avec Node.js. Pour ce faire, un

repository GitHub a été créé et le projet initialisé et poussé sur ledit repo. Afin de pouvoir travailler dessus sur votre machine, vous devez la configurer afin que celle-ci puisse exécuter le projet correctement.

Aussi, avez-vous tout d'abord commencer par installer l'outil Node.js sur votre machine ainsi que son gestionnaire de packages npm. Jusque là tout allait pour le mieux et aucun problème n'avait fait surface mais lorsque vous vous décidez de passer à l'installation des dépendances du projet, vous vous hurtez à de nombreux problèmes d'incompatibilité de versions des packages qui sont devenus obselètes et non pris en charge par votre version actuelle de npm. Là vous vous retrouvez coincer et incapable de contribuer au projet.

Décidé à ne pas vous avouer vaincu, vous entreprenez des recherches sur d'éventuels moyens de faciliter la création de l'environnement de travail sur chaque machine impliquée dans un projet de développement et tombez sur l'outil Docker. Lors de vos lectures sur ledit outil, vous apprenez qu'il permet de créer une image dite conteneur qui inclut l'intégralité de votre environnement de développement (Node.js, base de données, bibliothèques). Une fois cela fait, vous n'avez plus qu'à la partager et à l'exécuter sur n'importe quelle machine sur laquelle Docker est installé, pour garantir un environnement de développement cohérent et préconfiguré pour votre équipe.

Introduction

Docker a révolutionné la façon dont les applications sont développées, déployées et gérées. En tirant parti de la technologie de conteneurisation, Docker vous permet de regrouper votre application avec toutes ses dépendances dans une unité légère et portable appelée conteneur. Cette approche conteneurisée offre de nombreux avantages, notamment :

- **Cohérence et isolation améliorées :** les conteneurs s'exécutent indépendamment du système hôte et les uns des autres, garantissant un comportement cohérent dans tous les environnements.
- **Déploiement simplifié :** les conteneurs Docker sont autonomes, ce qui facilite le déploiement dans différents environnements.
- **Cycles de développement plus rapides :** les conteneurs tournent rapidement, ce qui permet des itérations de développement rapides.
- **Consommation réduite de ressources :** les conteneurs sont légers et partagent le noyau du système d'exploitation hôte, ce qui les rend économes en ressources.

Guide d'installation : configuration de votre environnement Docker

Conditions préalables

- Un ordinateur avec un système d'exploitation 64 bits (Windows 10 ou version ultérieure, macOS 10.10 ou version ultérieure, Linux)
- Capacités de virtualisation activées sur votre système (paramètres BIOS/UEFI)

Installation sous Windows, macOS et Linux

Des instructions détaillées pour l'installation de Docker sur chaque plate-forme peuvent être trouvées dans la documentation officielle de Docker. Voici un bref aperçu :

1. **Téléchargez le programme d'installation :** accédez à https://docs.docker.com/engine/install/ et téléchargez le programme d'installation approprié pour votre système d'exploitation.
2. **Exécutez le programme d'installation :** suivez les instructions à l'écran pour terminer l'installation.
3. **Vérifiez l'installation :** ouvrez un terminal ou une invite de commande et tapez « version docker ». En cas de succès, vous verrez la version de Docker installée.

Concepts clés de Docker

Conteneurs vs machines virtuelles

Même si les conteneurs et les machines virtuelles (VM) offrent un moyen d'isoler les applications, leur approche diffère. Les machines virtuelles virtualisent l'intégralité du système d'exploitation, tandis que les conteneurs partagent le noyau hôte, ce qui les rend plus légers et efficaces.

Images Docker : le modèle pour les conteneurs

Considérez une image Docker comme un modèle pour créer un conteneur. Il contient toutes les instructions nécessaires pour configurer l'environnement du conteneur, y compris le système d'exploitation, le code de l'application, les bibliothèques et les paramètres.

Dockerfile : définir la configuration de votre conteneur

Un Dockerfile est un fichier texte qui spécifie les instructions de création d'une image Docker. Il définit le système d'exploitation de base, installe les dépendances, copie le code de votre application et configure l'environnement d'exécution.

Structure

1. **FROM** : Spécifie l'image de base dont hériter (par exemple, Ubuntu, Python).
2. **WORKDIR** : définit le répertoire de travail pour les commandes suivantes.
3. **COPY** : copie les fichiers ou répertoires de la machine hôte dans le conteneur.
4. **RUN** : exécute les commandes dans le conteneur pendant le processus de construction (par exemple, l'installation des dépendances).
5. **CMD** : définit la commande à exécuter au démarrage du conteneur.
6. **EXPOSE** : expose les ports du conteneur pour un accès externe.

Exemple

```
FROM python:3.9

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

CMD ["python", "app.py"]

EXPOSE 5000
```

Ce Dockerfile crée une image basée sur Python 3.9, installe les dépendances, copie le code de l'application et démarre l'application à l'aide de « app.py » lorsque le conteneur s'exécute. Le port 5000 est exposé pour un accès externe.

Volumes Docker : stockage de données persistant pour les conteneurs

Les conteneurs sont éphémères par nature. Toutes les données créées dans un conteneur sont perdues lorsque le conteneur s'arrête. Les volumes Docker permettent de conserver les données en dehors du conteneur, garantissant ainsi leur disponibilité même après le redémarrage du conteneur.

Avantages

- **Persistance :** les données écrites sur les volumes persistent même après l'arrêt ou le redémarrage des conteneurs, garantissant ainsi la continuité des applications.
- **Partage :** les volumes peuvent être montés simultanément dans plusieurs conteneurs, permettant le partage de données entre les services.
- **Flexibilité :** les volumes sont indépendants des cycles de vie des conteneurs, ce qui simplifie la gestion et les mises à jour des conteneurs.
- **Sauvegarde et migration :** les volumes peuvent être facilement sauvegardés ou migrés entre les hôtes Docker, facilitant ainsi la reprise après sinistre et la portabilité des applications.

Types de volumes

- Volumes locaux (par défaut) : volumes stockés sur la machine hôte Docker dans un répertoire dédié (généralement /var/lib/docker/volumes). Idéal pour le développement et les tests.
- Volumes liés : montez un répertoire ou un fichier de la machine hôte directement dans un conteneur. Utile pour accéder aux données existantes sur l'hôte ou configurer des applications.

Commandes

1. **docker volume create <volume_name>** : crée un nouveau volume nommé.
2. **docker volume ls** : répertorie tous les volumes.
3. **docker volume inspect <volume_name>** : affiche des informations détaillées sur un volume.
4. **docker volume prune** : supprime les volumes inutilisés.
5. **docker volume rm <volume_name>** : Supprime un volume (à utiliser avec prudence, les données sont perdues).

Monter des volumes

Effectué lors de la création du conteneur à l'aide de l'indicateur -v ou --mount : * **<host_path>:<container_path>** : monte le répertoire hôte sur <host_path> sur le chemin du conteneur sur <container_path> * **<volume_name>:<container_path>** : monte le volume nommé <volume_name> sur le chemin du conteneur à <container_path>

Exemple

```
docker volume create my-data  # Créer un volume nommé

docker run -d -v my-data:/app/data my-app # Exécuter un conteneur avec le volume
```

Cela crée un volume nommé my-data, puis exécute le conteneur my-app avec le volume monté sur /app/data à l'intérieur du conteneur. Toutes les données écrites dans ce répertoire persistent dans le volume et sont accessibles lors des redémarrages du conteneur ou même lorsqu'elles sont utilisées avec différents conteneurs.

L'essentiel de Docker

Création d'images Docker

Une fois que vous avez défini un fichier Docker, vous pouvez utiliser la commande

```
docker build [OPTIONS] PATH | URL | -
```

pour créer une image Docker à partir de vos instructions.

Exécution de conteneurs Docker

Utilisez la commande

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

pour créer et exécuter une instance de conteneur à partir d'une image Docker spécifique. Vous pouvez éventuellement spécifier des paramètres supplémentaires tels que des ports, des volumes et des variables d'environnement.

Gestion des objets Docker (images, conteneurs, volumes) :

Docker fournit un ensemble de commandes pour gérer les objets Docker :

1. **docker images** : répertorie toutes les images Docker sur votre système.
2. **docker ps** : répertorie tous les conteneurs en cours d'exécution.
3. **docker stop <container_name>** : arrête un conteneur en cours d'exécution.
4. **docker rm <container_name>** : supprime un conteneur arrêté.
5. **docker volume ls** : répertorie tous les volumes Docker.
6. **docker volume prune** : supprimez les volumes Docker inutilisés.

Mise en réseau entre conteneurs

Les conteneurs Docker peuvent être configurés pour communiquer entre eux à l'aide d'un réseau privé. Cela vous permet de créer des applications multi-conteneurs où les services peuvent interagir de manière transparente.

Docker propose plusieurs modes réseau : * **Pont (par défaut)** : les conteneurs se connectent à un réseau virtuel et peuvent communiquer entre eux à l'aide d'adresses IP. * **Hôte** : les conteneurs partagent la pile réseau de la machine hôte. * **Superposition (mode Swarm)** : utilisé pour la communication entre les conteneurs sur plusieurs nœuds Swarm. * **Personnalisé** : permet de créer des réseaux définis par l'utilisateur avec des configurations spécifiques.

Exemple :

Dans l'exemple `docker-compose.yml`, un réseau personnalisé `my-network` est défini. Cela permet aux services « Web » et « base de données » de communiquer entre eux en utilisant des noms de conteneurs ou des noms d'hôtes au sein du réseau.

Docker Compose : simplifier les applications multi-conteneurs

La gestion d'applications complexes avec plusieurs conteneurs peut devenir fastidieuse. Docker Compose vient à la rescousse en vous permettant de définir et d'exécuter des applications multi-conteneurs avec un seul fichier YAML. Docker Compose simplifie la découverte de services, la mise en réseau et la mise à l'échelle au sein de votre application multi-conteneurs.

Structure

```
version: '3.8'

services:
  web:
    build: .  # Construit l'image à partir du Dockerfile du répertoire courant
    ports:
      - "5000:5000"  # Mappe le port de conteneur 5000 au port hôte 5000
    volumes:
      - ./app:/app  # Construit l'image à partir du Dockerfile du répertoire courant

  database:
    image: postgres  # Utilise une image Postgres prédéfinie

networks:
  my-network:  # Définit un réseau personnalisé pour les services à connecter
    external: true  #  Connecte le réseau au réseau hôte

volumes:
  data:  # Définit un volume nommé pour les données persistantes
```

Exemple

Ce `docker-compose.yml` définit deux services : `web` (construit à partir d'un Dockerfile) et `database` (en utilisant une image prédéfinie). Les deux sont connectés au réseau personnalisé « mon-réseau ». Le service « web » monte également le répertoire du code de l'application sur l'hôte pour faciliter le développement.

Bonnes pratiques

Voici quelques bonnes pratiques Docker à prendre en compte pour une conteneurisation efficace et sécurisée :

Bonnes pratiques Dockerfile :

1. Images de base :
    - Utilisez les images Docker officielles de Docker Hub autant que possible. Ils sont bien entretenus, testés et sécurisés.
    - Choisissez une image de base fine et contenant uniquement ce dont votre application a besoin. Évitez les images de base gonflées comme Ubuntu:latest. Envisagez des alternatives comme Alpine Linux pour des tailles d'image plus petites.
2. Constructions en plusieurs étapes :
    - Utilisez des constructions en plusieurs étapes pour créer des images plus petites et plus efficaces. La première étape peut contenir toutes les dépendances de build nécessaires à la création de l'application, tandis que la dernière étape peut être une image minimale contenant uniquement le code de l'application et l'environnement d'exécution.

3. Réduire les calques :
    - Gardez le nombre de couches dans votre Dockerfile au minimum. Chaque instruction `RUN` crée un nouveau calque. Combinez plusieurs commandes en une seule instruction « RUN » lorsque cela est possible pour réduire la taille de l'image.
4. Utilisez `.dockerignore` :
    - Créez un fichier « .dockerignore » pour exclure la copie des fichiers et répertoires inutiles dans l'image pendant le processus de construction. Cela réduit la taille de l'image et le temps de construction.
5. Spécifiez l'utilisateur :
    - N'exécutez pas votre application en tant que root dans le conteneur. Choisissez un utilisateur non privilégié pour une meilleure sécurité.
6. Versions de broches :
    - Spécifiez toujours les versions exactes des images de base et des dépendances dans votre Dockerfile. Cela garantit la reproductibilité et évite les changements inattendus.

Meilleures pratiques générales :

- **Tomes** :
    - Utilisez des volumes pour stocker les données persistantes qui doivent survivre aux redémarrages des conteneurs. Cela permet de séparer les données de votre application de l'image du conteneur elle-même.
- **Réseaux** :
    - Tirez parti des réseaux Docker pour gérer la communication entre les conteneurs. Définissez des réseaux personnalisés pour votre application afin de l'isoler des autres conteneurs.
- **Sécurité** :
    - Minimisez la surface d'attaque en exposant uniquement les ports nécessaires sur vos conteneurs.
    - Analysez régulièrement vos images Docker à la recherche de vulnérabilités à l'aide d'outils tels que « Docker Scan ».
    - Pensez à utiliser des outils de gestion des secrets pour stocker des informations sensibles en dehors de l'image du conteneur.
- **Enregistrement** :
    - Configurez votre application pour qu'elle se connecte à la sortie standard (stdout) ou à l'erreur standard (stderr) pour une collecte et une surveillance efficaces des journaux.
- **Essai** :
    - Implémentez des tests automatisés pour vos fichiers Docker pour vous assurer qu'ils sont construits de manière cohérente et produisent l'image attendue.
- **Documentation** :
    - Documentez vos Dockerfiles et vos configurations de déploiement pour une meilleure

maintenabilité et une meilleure compréhension.

En suivant ces bonnes pratiques, vous pouvez créer des images et des applications Docker sécurisées, efficaces et maintenables. N'oubliez pas qu'il ne s'agit là que de quelques-unes des nombreuses bonnes pratiques disponibles. Il existe des considérations supplémentaires en fonction de votre cas d'utilisation spécifique.

Outils Docker : rationaliser les flux de travail des conteneurs

Docker lui-même est un outil essentiel pour créer, exécuter et gérer des conteneurs individuels. Cependant, l'écosystème Docker offre une variété d'outils complémentaires pour améliorer votre expérience de conteneurisation :

Docker Hub : un registre d'images public

Comme mentionné précédemment, Docker Hub sert de registre public pour le partage et la découverte d'images Docker. Il vous permet de :

1. Parcourez et téléchargez des images prédéfinies pour diverses applications et frameworks, ce qui vous fait gagner du temps de développement.
2. Transférez vos propres images personnalisées vers le Hub pour les partager avec d'autres ou au sein de votre équipe.
3. Explorez une vaste collection d'images créées par la communauté, favorisant la collaboration et l'innovation.

Docker Swarm : orchestrer des applications conteneurisées

Docker Swarm est un outil de clustering natif permettant de gérer et de faire évoluer des applications multi-conteneurs. Il facilite :

1. Déploiement et mise à l'échelle de services conteneurisés sur un cluster d'hôtes Docker.
2. Équilibrage de charge du trafic sur plusieurs instances de conteneur pour un service.
3. Capacités d'auto-guérison pour remplacer automatiquement les conteneurs malsains au sein de l'essaim.

Alors que Docker fournit des fonctionnalités de base pour exécuter des conteneurs individuels, Docker Swarm vous permet de gérer des applications conteneurisées complexes et distribuées.

Concepts

- **Services :** définissez le nombre de réplicas de conteneurs à exécuter et comment les mettre à l'échelle.

- **Piles :** regroupez les services et les configurations associés pour le déploiement.
- **Superpositions :** mettez à jour les déploiements existants avec de nouvelles configurations.

Exemple

Imaginez une application Web avec plusieurs conteneurs de serveur Web et un conteneur de base de données. À l'aide de Docker Swarm, vous pouvez définir un service pour chaque type de conteneur avec les réplicas souhaités. L'essaim gérera le cycle de vie des conteneurs et assurera une haute disponibilité en cas de panne.

Outils Docker supplémentaires : l'écosystème plus large

Au-delà de ces outils de base, l'écosystème Docker propose une variété d'outils supplémentaires pour répondre à des besoins spécifiques :

1. **Docker BuildKit** : un générateur d'images avancé axé sur des temps de construction plus rapides et des mécanismes de mise en cache améliorés.
2. **Docker Registry** : une alternative d'entreprise à Docker Hub, offrant des référentiels d'images privés avec une sécurité et des contrôles d'accès améliorés.
3. **Docker Desktop (pour Windows et macOS)** : une interface graphique conviviale pour gérer Docker sur votre bureau, simplifiant la création de conteneurs, l'interaction et la gestion des ressources.

En exploitant efficacement ces outils, vous pouvez créer un workflow de développement et de déploiement conteneurisé robuste et efficace.

Conclusion : les avantages des workflows dockerisés

Docker est devenu un outil indispensable pour les développeurs et les équipes opérationnelles. En tirant parti de la conteneurisation, Docker rationalise les flux de développement, simplifie les processus de déploiement et favorise des applications cohérentes et portables. Que vous soyez un développeur Web, un administrateur système ou simplement curieux de connaître les pratiques modernes de développement d'applications, Docker propose un ensemble d'outils puissants pour améliorer votre flux de travail et vous permettre de créer et de déployer des applications avec une plus grande efficacité.

Annexe : Ressources pour un apprentissage ultérieur

- Documentation Docker : https://docs.docker.com/engine/install/
- Docker Hub : https://hub.docker.com/

- Une visite guidée de Docker : https://docs.docker.com/get-started/
- Documentation Docker Compose : https://docs.docker.com/compose/

Cette documentation fournit une compréhension fondamentale des concepts et fonctionnalités de Docker. Les ressources fournies proposent une exploration plus approfondie et des didacticiels pour vous doter des compétences nécessaires pour exploiter efficacement Docker dans vos processus de développement et de déploiement.